

Programación 2

**Tecnatura en Desarrollo de Aplicaciones
Informáticas**

Colecciones

Límites de los Arreglos

— — —

Que problema tenemos con los arreglos?

Colecciones Dinámicas

ArrayList

ArrayList es una clase en Java import **java.util.ArrayList;**

Proporciona una estructura de datos dinámica para almacenar elementos de manera consecutiva.

A diferencia de los arrays estáticos, los ArrayLists pueden crecer o disminuir automáticamente según sea necesario.

ArrayList

— — —

```
ArrayList<String> nombres = new ArrayList<>();
```

```
nombres.add("Alice");
```

```
nombres.add("Bob");
```

```
nombres.add("Charlie");
```

ArrayList

```
String primerNombre = nombres.get(0); // Alice
```

```
nombres.set(1, "Robert"); //Modifica el valor de la posición 1
```

```
nombres.add(1,"Juan"); //agrega un elemento en la posicion 1 y  
realiza un corrimiento de los restantes
```

```
nombres.remove(2); // Elimina "Robert" (corrimiento)
```

```
int tamaño = nombres.size(); // 3
```

```
int indice = nombres.indexOf("Alice"); // 0
```

ArrayList

```
for (int i =0; i<nombres.size();i++) {  
    System.out.println(nombres.get(i));  
}
```


Conceptos

Resumen de Conceptos - Agenda

palabra reservada **this**

Paquetes en java

Modificadores de Acceso

palabra reservada **super**

Atributos y Métodos de Clase

palabra clave **final**

this

this

Es una palabra clave que referencia al **objeto actual**

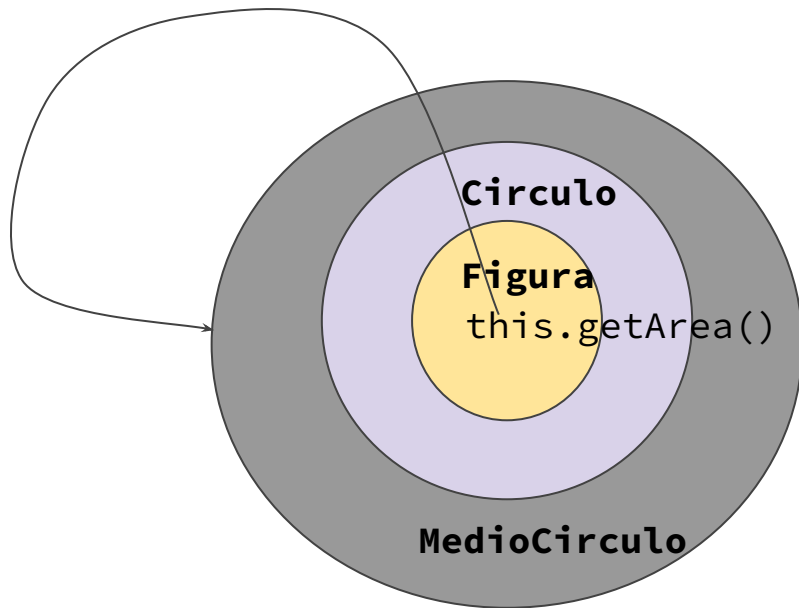


Figura → Circulo → MedioCirculo

MedioCirculo Hereda de Circulo y Circulo de Figura.

Si creamos una instancia de MedioCirculo y le enviamos un mensaje el cual termina con la ejecución de un metodo que esta en figura y usa **this**, El mensaje enviado es al OBJETO ACTUAL es decir a la instancia de MedioCirculo (es como si volviera al inicio)

this

Tiene varias utilidades:



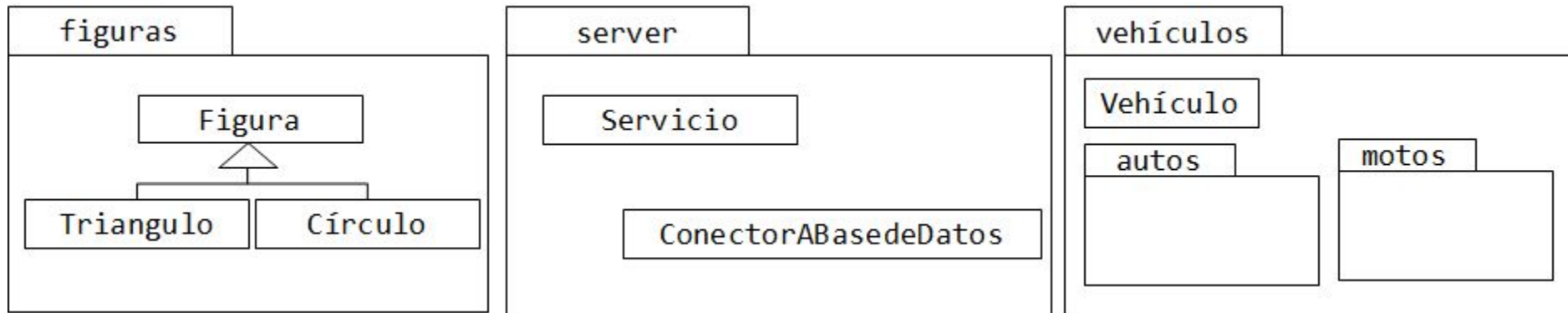
- Poder manipular el objeto actual (por ej., pasarlo como parámetro)
 - `System.out.println(this)`
- Diferenciar entre variables/parámetros y atributos locales
 - `this.nombre = nombre; //nombre es var local del método y this.nombre hace referencia al atributo de la clase`
- Reutilizar un constructor existente
 - `public Persona(String nombre, int dni){`
 - `this(dni);`
 - `this.nombre = nombre;`

Paquetes en Java

Paquetes

Proveen una forma de organizar el código fuente de nuestra aplicación

Define un espacio de nombres para nuestras clases



Paquetes

Escritos en minúscula.

Para acceder a cada nivel se utiliza el "."

Por ejemplo, vehículos.autos.Camioneta

Se mapean directamente a carpetas del sistema de archivos

Declaración de Paquetes

— — —

```
package tp3.ejercicio1;
```

```
public class Persona {....
```

Se dice que la clase Persona pertenece al paquete “ejercicio1” dentro del paquete “tp3”

Importar Paquetes

— — —

```
package tp3.ejercicio1;
```

// Se colocan los imports uno debajo del otro y luego de la declaración del paquete.

```
import java.util.Vector;;//Importar una clase de la plataforma Java
```

```
import tp3.ejercicio2.Electrodomestico;
```

```
import java.io.*; //Importar TODAS las clases en el paquete.
```

```
public class Persona{
```

```
    }
```

```
}
```

Modificadores de Acceso

Modificadores de Acceso

Java provee distintos modificadores de Acceso que pueden usarse en Atributos o Métodos

```
protected int edad;
```

```
private void ordenarSecuencia() {....}
```

NOTA: NO definir nunca atributos públicos porque esto viola el encapsulamiento de un Objeto



Modificadores de Acceso

Para Atributos y Métodos:

	Misma clase	Mismo paquete	Clase hija (Herencia)	Otro paquete
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
Sin modificador	✓	✓	✗	✗
private	✓	✗	✗	✗

super

super

La palabra clave `super` permite referenciar explícitamente a los métodos y atributos que son parte de la clase superior

Continuando con el Ejemplo del `MedioCirculo`

```
public double getArea() {  
    return super.getArea()/2;  
}  
// El Area de un MedioCirculo es la mitad del Area  
del Circulo (de quien hereda) No se repite la fomra de  
calculo se re usa el comportamiento
```

super

De forma similar a this, se utiliza para:

- Utilizar de forma explícita **métodos** y **atributos** de la clase superior (en particular, es útil cuando deseamos invocar un método de la clase padre desde un método sobrescrito)
- Utilizar **constructores** de la clase superior (muchas veces es obligatorio)

```
public Alumno(int dni, int legajo){  
    super(dni);  
    this.legajo=legajo;  
}
```


super

— — —

- A veces, es deseable distinguir entre llamadas a atributos o métodos de la clase superior
- En otros casos, es necesario para evitar errores de código.
- En general, **se desaconseja** utilizar atributos de forma directa mediante `super`

Atributos y Métodos de Clase

Atributo de Clase

Un **único valor** para un atributo de una clase (es el mismo para todos los objetos creados, cero o más)

Por ejemplo un contador de Personas Creadas

```
static int contador;
```

Atributo de Clase

La palabra `static` en la definición de atributo nos dice que es un atributo de clase

Para los objetos es un atributo más, la diferencia es que es la MISMA variable para todos los objetos, si un objeto cambia el valor, este cambia para TODOS los objetos;

```
contador = 0 ; // re inicia el contador para todos
```

```
contador ++; //incrementa en uno el contador para
```

TODOS

Atributos static

Son variables asociadas con la clase y **NO** con instancias particulares.

Para acceder a un atributo de clase NO es necesario crear instancias de la clase. Pueden ser accedidas usando el nombre de la clase y sin haber creado instancias.

Son GLOBALES para la clase.

Métodos de Clase

Es un método que no se encuentra asociado a los objetos de la clase

Un método que pueda ser invocado incluso cuando no existan objetos

Por Ejemplo

`Math.random()` // `random` es un método de clase de la clase `Math`, no necesitamos crear un objeto para invocarlo

Métodos de Clase

Definición de un método de clase

```
public static int getContador() {  
    return contador;  
}
```

Se puede Acceder usando `Persona.getContador();`

Métodos de clase

NO se pueden invocar métodos no **static** dentro de un método `static`. (cuando se invoca un método `static` hay que pensar que no existe una instancia)

La reversa si es posible.

Es posible invocar un método estático desde la propia clase que lo define, sin ningún objeto.

CUIDADO static



Las variables locales **NO** pueden ser static.

Solo los **atributos** pueden ser **static**.

Si el atributo static es de un tipo primitivo y no es inicializado, toma el valor standard inicial para su tipo.

```
int    0
float, double  0.0
boolean false
```

Si el atributo static es la referencia a un objeto, toma el valor de null.

final

final

La palabra clave final se utiliza para definir “constantes”

En un **atributo** es un valor que nunca cambia

En una **clase** es una clase que no puede ser extendida (no se puede heredar de ella)

En un **método** es un método que no puede ser redefinido (la clase que extienda no puede declararlo nuevamente)

final - Atributo

— — —

final int MAYORDEEDA = 23; //Define en la clase persona una constante para la mayoría de Edad. Por convención, La constantes siempre van con mayúsculas

Ojo, no tiene sentido una constante por instancia!!!! por eso se usa

static final int MAYORDEEDA = 23; //Solo se conoce dentro de la clase

Si queremos que todos conozcan la constante

public static final int MAYORDEEDAR = 23;

En este último caso se accede a la constante → **Persona.MAYORDEEDAR**

final - Clase

La clase no puede ser extendida

```
public final class Persona { . . . }
```

No se puede heredar de persona

```
public class Alumno extends Persona { // ERROR
```

final - Método

El método no se puede sobre escribir:

// En Figura

```
public final String getNombre() { . . . }
```

Nadie puede cambiarlo

// En Circulo

```
public String getNombre() { // ERROR
```

La Palabra Clave final - RESUMEN

— — Atributo final

No se puede cambiar su valor.

Método final

No se puede sobre-escribir el método.

Clase final

No se puede extender la clase (no se puede crear una sub-clase).